UNIVERSITATEA DE VEST DIN TIMIŞOARA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENŢĂ: Informatică

LUCRARE DE LICENŢĂ

COORDONATOR:
Prof. Dr. Daniela Zaharie

ABSOLVENT:
Negrea Alexandru Cristian

COLABORATOR:
Drd. Alexandru Meca

TIMIŞOARA
2021

UNIVERSITATEA DE VEST DIN TIMIŞOARA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENŢĂ: Informatică

Dezvoltarea aplicațiilor WebGIS prin utilizarea bazelor de date cu suport spațial.

COORDONATOR:
Prof. Dr. Daniela Zaharie

COLABORATOR:
Drd. Alexandru Meca

ABSOLVENT:
Negrea Alexandru Cristian

TIMIŞOARA
2021

Abstract

The aim of this bachelor's thesis is to design a software application (just the back-end) to be used as a tool for spatial data processing, extracting information about given locations etc. This bachelor's thesis is preponderantly practical because it shows how a software application is made to solve a problem. Advanced characteristics of programming languages - such as Java, and the popular framework Spring Boot - and the libraries available in Java will be used, as well as programming techniques. The thesis will provide functionality for a web application(for a front-end app) to utilize it's facilities such as : extract data about a given institution type like : medical, public etc, get specific institution by a given name/id, extract locations from given zone with a given radius, the closest location from a given point etc.

Cuprins

Introducere

Asa cum se poate observa si in titlul lucrarii de licenta tema este dezvoltarea aplicațiilor WebGIS prin utilizarea bazelor de date cu suport spațial. Obiectivul principal al lucrarii de licenta pentru tema specificata mai sus il reprezinta utilizarea de tehnologii specifice si implementarea unei aplicatii software pentru prelucrarea datelor spatiale. Aplicatia este un server-based Web Feature Service (WFS) care manipuleaza informatii de tipul geografic cum ar fi : puncte, linii, poligoane etc, aceasta permitand operatii de tip query basic CRUD pe date spatiale prin intermediul Web-ului.

Aceasta aplicatie, care va fi bazata pe microservicii cu comunicare decuplată între componente(micro serviciile) spațiale prin implementare integrala pe infrastructură cloud AWS, se va ocupa cu returnarea de locatii de interes pentru utilizatori dintr-un oras anume pe ideea de “SmartCity“ unde acestia pot primi informatii precum: cele mai apropiate institutii medicale, spatii publice, retele de transport etc. De exemplu un utilizator doreste sa indentifice locuri de munca care sunt cele mai apropiate de locuinta sa sau locuinte care sunt foarte aproape de locul de munca. Aceste locatii vor putea fi vizibile prin intermediul API-ului de la “Google“ (Google Maps) pentru a avea o vedere cat mai detaliata si amanuntita a distantelor si a imprejurimilor.

Capitolul 1

Descrierea problematicii

In acest capitol se va discuta despre descrierea problematicii temei alese, solutii si abordari similare unde se vor exemplifica caracteristici si diferente dintre aplicatiile deja existente si solutia creata de mine, descrierea instrumentelor pentru rezolvarea problematicii etc.

1.1 Descrierea problematicii temei alese

Asa cum s-a precizat si in partea de introducere a lucrarii alese, principalul scop al aplicatiei il reprezinta dezvoltarea aplicatiilor WebGIS prin utilizarea bazelor de date cu suport spatial ce utilizeaza WFS (Web Feature Service).

- **WFS** reprezinta o schimbare a modului în care informatiile geografice sunt create, modificate si schimbate pe internet. In loc sa partajeze informatii geografice la nivel de fisier folosind Protocolul de transfer de fisiere (FTP), de exemplu, WFS ofera acces direct cu informatii fine la informatii geografice la nivel de caracteristica si proprietate a caracteristicii.

Acesta functioneaza prin trimiterea de cereri ce contin descrieri despre query-ul dorit cat si tipul de transformare de informatii ce trebuiesc efectuate. Request-ul trebuie sa fie generat pe partea de client (Web Client) si trimisa catre aplicata server-based care este pana la urma un server API. Acesta va citii datele si le va executa corespunzator returnand rezultatul dorit.

WebGIS este o forma avansata de un sistem geospatial de informatii (cunoscut ca Sistemul Geografic de Informatii) care este disponibil pe platforma web.Acest schimb de informatii se realizeaza intre intre un server GIS (Geospatial Information System) si un client care este reprezentat de un web browser, aplicatie mobila sau aplicatie de desktop.

- **GIS:** este un sistem computer-based geografic de informatii folosit pentru colectarea, stocarea, manipularea si analizarea datelor spatiale, cum ar fi datele care pot fi referentiate la o locatie particulara pe pamant.

- **Spatial data:** cunoscut ca si geospatial data, este o informatie despre obiect fizic care poate fi reprezentat de valori numerice intr-un sistem de coordonate

ce este reprezentat de: numere ce reprezinta latitudine si longitudinea in format 2D.

Obiectivul principal al acesteia il reprezinta crearea unei aplicatii server based bazate pe microservicii unde prin legarea acesteia de un web client, utilizatorul poate sa realizeze multiple operatiuni similare cu aplicatia celor de la Google, **Google Maps**. Comunicarea dintre cele doua se va realiza prin request-uri HTTP. Prin header folosind "GET", dar si "POST" pentru trimiterea de obiecte in format JSON.

Aplicatia implementata este bazata pe microservicii a caror avantaje si functionlitati vor fi descrise in urmatoarele capitole. Prin intermediul acesteia, un utilizator prin cadrul unei interfete web/aplicatii de telefon, ar putea obtine informatii despre o locatie specifica de pe harta, sa caute o anumita locatie in functie de numele acesteia, sa calcuze distanta dintre doua locatii date, sa obtina cea mai apropiata locatie de acesta in functie de institutia dorita, sa obtina toate locatiile de acelasi tip din acea zona sau sa obtina toate tipurile de locatii din raza data. Aceste functionalitati vor fi mai detaliate in capitolul 3, **Facilitati Aplicatie**. In esenta, aceasta aplicatie este creata pentru implementarea ideei de "SmartCity".

SmartCity: este un framework, predominant alcatuit din Informatii si Tehnologii de Comunicare (ICT: Information and Communication Technologies), pentru a dezvolta, implementa si promova practici de dezvoltare durabila pentru a aborda provocarile în crestere ale urbanizarii. O mare parte a framework-ului ICT este esential un network inteligent de obiecte conectate si masinarii (cunoscute ca si orase digitale) care transmit date folosind tehnologii wireless si cloud.

Aplicatiile IoT bazate pe cloud primesc, analizeaza si gestioneaza datele în timp real pentru a ajuta municipalitatile, intreprinderile si cetatenii sa ia decizii mai bune care imbunatatesc calitatea vietii. Dupa mai multe statistici s-au identificat 4 categorii de smart-city care sunt cele mai utilizate, acesate fiind:

- **Essential Services Model:** Orasele din categoria Essential Services Model se caracterizeaza prin utilizarea retelelor mobile în programele lor de gestionare a situatiilor de urgenta si prin serviciile lor digitale de asistenta medicala. Aceste orașe, care ar putea avea deja infrastructuri de comunicatii bune, prefera sa-si puna banii in cateva programe de oras inteligent bine alese. Exemplele includ Tokyo si Copenhaga.

- **Smart Transportation Model:** Orașele Smart Transportation Model cuprind cele dens populate si care se confrunta cu probleme de deplasare a marfurilor si a persoanelor in oras. Orasele din acest grup subliniaza initiativele de control al congestionarii urbane - prin intermediul transportului public inteligent, partajarii masinilor si / sau autoturismelor - precum si utilizarii tehnologiilor informatiei si comunicatiilor. Singapore si Dubai sunt incluse în aceasta categorie.

- **Broad Spectrum Model:** Orasele care intra sub incidenta Broad Spectrum Model subliniaza serviciile urbane, cum ar fi gestionarea apei, a apelor uzate si a deseurilor si cauta solutii tehnologice pentru controlul poluarii. De asemenea, acestea se caracterizeaza printr-un nivel ridicat de participare civica. Exemplele includ Barcelona, Vancouver si Bejing.

- **Business Ecosystem Model:** Urmareste sa utilizeze potentialul tehnologiilor informatiei si comunicatiilor pentru a incepe activitatea economica. Include orase care pun accentul pe formarea in competente digitale ca acompaniament necesar pentru a crea o forta de munca instruita si care vizeaza sa incurajeze intreprinderile de inalta tehnologie. Amsterdam, Edinburgh si Cape Town sunt exemple.

Implicarea WebGIS in crearea ideii de Smart City il reprezinta un sistem centralizat de informatii bazat pe GIS ofera un cadru IT pentru intretinerea si implementarea datelor si aplicatiilor de-a lungul fiecarui aspect al ciclului de viata al dezvoltarii orasului.

Exemple de aplicatii pentru Smart Cities:

- **Selecția site-ului și achiziționarea terenurilor:** GIS poate combina si integra diferite tipuri de informatii pentru a ajuta la luarea unor decizii mai bune si de asemenea ofera, instrumente de vizualizare de inalta calitate a imprejurimilor care pot imbunatatii luarea de decizii. De exemplu: o persoana doreste sa achizitioneze un apartament si ar vrea intr-o zona unde este aproape de locul de munca sau alte puncte de interes. Acesta folosind asemenea aplicatii va putea vizualiza in detaliu zone care sunt in interesul acestuia.

- **Planificare, proiectare si vizualizare:** Geodesignul va fi cadrul cheie pentru conceptualizarea si planificarea oraselor inteligente; va ajuta in fiecare etapa, de la conceptualizarea proiectului la analiza amplasamentului, specificatiile de proiectare, participarea si colaborarea partilor interesate, crearea proiectelor, simularea si evaluarea.

- **Vanzari si marketing:** Cu GIS, dezvoltatorii orasului pot castiga potentialele afaceri prin crearea de instrumente informative de vanzari si rapoarte de marketing care evidentiaza potentialul economic al unei noi locatii sau al dezvoltarii viitoare.

De asemenea folosind acest GeoRESTAPI se mai folosii: **Open Data** care sunt date ce pot fi folosite în mod liber, refolosite și redistribuite de catre oricine si în orice scop, supuse cel mult la necesitatea atribuirii in conditii identice. Astfel se vor putea

face previzualizare a datelor spatiale tinand cont de standardele OGC si prin API-ul aplicatii se vor pune la dispozitie ca un corp platform in sprijinul de dezvoltare de aplicatii care sa consume aceste date.

- **OGC:** este o organizatie internationala de standardizare a consensului voluntar, ce a luat nastere în 1994. In OGC, peste 500 de organizatii comerciale, guvernamentale, nonprofit si de cercetare din intreaga lume colaboreaza intr-un proces de consens care încurajeaza dezvoltarea si implementarea standardelor deschise pentru continutul geospațial si servicii, senzori web si Internetul obiectelor (Iot - Internet of Things), prelucrarea datelor GIS si partajarea acestora. Aceasta organizatie a standardizat mai multe elemente GIS, unele dintre ele fiind:
  - **SRID:** o identificare pentru sistemul de coordonare spatial
  - **WMS:** Web Map Service ofera imagini de pe harta, aceasta reprezentat aplicatiile Web ce utilizeaza server-based applications ce implementeaza serviciile WebGIS, cateva exemple ar fi: Google Maps, Bing Maps etc.
  - **WFS:** Web Feature Service ce sunt folosite pentru recuperarea sau modificarea descrierilor caracteristicilor
  - etc

1.2 Soluții și abordări similare

Solutiile existente pe piata sunt :

- **ArcGIS REST API** care este un REST API care ofera servicii geospatiale, mapare si administrative. Acesta poate fi folosit ca librarie pentru: iOS, Java, .Net etc. Acest REST Api este disponibil pentru o varietate de limbaje de programare cum ar fi : C# (impreuna cu framework-ul .Net), JavaScript, Python etc. Pe langa asta, ofera facilitati profesionale si optime cum ar fi : “Basemap and data layers“,“Geocoding“, “Data visualization“ si asa mai departe.
- **Bentley Map Enterprise** este un GIS 3D complet dotat cu capacitate puternica de editare a MicroStation. Este proiectat pentru a raspunde nevoilor unice ale organizatiilor care mapeaza, planificati, proiectati, construiti și operati infrastructura lumii. Are capacitatea de a lucra nativ cu toate cele comune surse de date spatiale precum PostGIS, Oracle, Microsoft SQL Server Spatial si multe altele. Bentley Map poate adăuga informatii semantice catre reteaua de realitate 3D. Harta Bentley ofera, de asemenea, un SDK complet pentru dezvoltarea personalizata Aplicatii GIS.

- **GeoMedia** este o platforma de gestionare GIS puternica, flexibila, care va permite sa agregati date dintr-o varietate de surse si sa le analizati la unison pentru a extrage informatii clare, care pot fi actionate. Ofera acces simultan la date geospatiale în aproape orice forma si le afiseaza intr-o singura vizualizare unitara a hartii pentru procesare, analiza, prezentare si partajare eficienta. Functionalitatea GeoMedia il face ideal pentru extragerea de informatii dintr-o serie de date care se schimba dinamic pentru a sprijini luarea de decizii în cunostinta de cauza, mai inteligenta.

- **GeoServer** este un server bazat pe Java care permite utilizatorilor să vizualizeze și să editeze date geospațiale. Folosind standardele deschise stabilite de Open Geospatial Consortium (OGC), *GeoServer* permite o mare flexibilitate în crearea hărților și partajarea datelor.La fel ca si GeoTools mentionat mai sus este "Use Free" si "Open Source". Pe langa asta, acest API folsoit impreuna cu "Open Layers" si "Leaflet".

Cateva caracteristici ale acestor solutii deja existente pe piata sunt urmatoarele:

- O caracteristica importanta comuna ale acestor solutii o reprezinta featurele ce le ofera tuturor clientiilor lor cum ar fi: fiabilitatea serviciilor oferite, fiind servicii care sunt in industrie de multi ani ofera un suport si stabilitate foarte buna pentru clientii acestora cat si un numar mare de feature ce le pot utiliza acestia

- Posibilitatea de a alege dintr-o gama larga de planuri de subscriptie, planul ce contine serviciile si feature-urile care sunt cele mai utilizate si importante pentru dezvoltarea aplicatiei tale

- Suport expert si customer service profesional care te poate ajuta imediat cu orice problema ai avea in utilizarea acestor servicii

- Performate exceptionale din partea serviciilor oferite

- etc

Totusi, cu toate ca aceste aplicatii sunt similare cu implementarea descrisa in lucrarea aceasta, sunt cateva diferente care merita sa fie explicate si exemplificate pentru a se putea observa o alta abordare in rezolvarea problemei folosind abordari diferite. Cateva dintre aceste diferente ar fi:

- Implementarea exemplificata foloseste librarii open source pentru manipularea datelor si returnarea acestora in functie de request-ul utilizatorului cum ar fi: GeoTools care este un o librarie open source ce iti ofera o multitudine de posibilitati de configurare si manipulare a datelor, de asemenea aceasta respecta standardele OGC pentru query-ul de date.

- Respectarea standardelor OGC astfel incat aplicatia poate fi folosita de mai multe aplicatii WMS care sa consume endpoint-urile API-ului creat fara a face modificari in ceea ce priveste primirea si transmiterea de date

- Utilizarea de baza de date open source PostgreSQL care foloseste ca extensie suplimentara PostGIS ce adauga suportul pentru manipularea de date spatiale si in acelasi timp respectand standardele OGC, astfel aplicatia nu este restrictionata de aceste standarde pentru manipularea de operatii CRUD atat in memorie folosind GeotTools cat si PostgreSQL cu extensia PostGIS

- O alta diferenta importanta, prin care aplicatia mea se diferentiaza de cele create de competitori, o reprezinta faptul ca fiind aplicatii cu o vechime mai mare folosesc o arhitectura de tipul **"Monolith"**, iar aplicatia exemplificata in lucrare va folosi microservicii care vor permite comunicarea cu device-uri multiple prin formatele standard OGC , dar si comunicarea si colectarea de date prin intermediul **"Internet of things"**.

  Aplicatia va fi alcatuita din 2 microservicii care vor avea functinalitati similare dar metode de implementare diferite. Unul din acestea va utiliza baza de date PostgreSQL cu extensia de geodate spatiale PostGIS, iar celalalt microserviciu va utiliza libraria "GeoTools". Cum aplicatia creata are la baza cele doua microservicii care folosesc diferite abordari pentru parsarea si manipularea datelor, majoritatea aplicatiilor existente pe piata folosesc arhitecturi de tip Monolith, care sunt de tip traditional, unde toate componentele acesteia sunt puse intr-un sigur loc ceea ce aduce la o multitudine de dezavantaje cum ar fi:

  - **Scalabilitate:** Cu cat aplicatia creste, va fi din ce in ce mai greu de mentenat, modulele ar putea avea conflicte in impartirea resurselor, iar la apartia unui bug/malfunction va fi dificil in indentificarea acestuia daca acesta va ajunge sa fie enterprise level grade application.
  - **Agilitate:** Cum se observa, in zilele noastre, cazurile de utilizare a businessurilor sunt extrem de agile, procesele de business continuă să evolueze. De exemplu, intr-o aplicatie bancara, continua sa apara noi reglementari. O aplicatie de tip Monolith se poate acomoda la aceste schimbari, dar la costul de reducere al agilitatii si al mentenantei deoarece si daca am schimba o mica parte in aplicatie, toata aplicatia trebuie sa fie verificata ca sa nu exista probleme de dependenta, trebuie sa fie reimpachetata si asamablata impreuna. Asadar se va reduce substantial productia iar, la un moment dat, procesul de mentenanta va fi extrem de dificil.
  - **Incalcarea principiului SRP:** "Single Responsability Principle" care spune ca: fiecare modul, clasa sau functie intr-un program ar trebui sa

aiba responsabilitatea de a se ocupa de o singura parte a aplicatiei si ar trebui sa incapsuleze acea parte de celelalte componente ale aplicatiei. Incalcand aceasta regula, implementarea unei noi functionalitati ar fii destul de dificila deoarece ar putea exista dependinte complexe si cum codul ar fi "spaghetti code" ar reduce productivitatea programatorilor enorm de mult.

Mai jos este descrisa structura generala unei aplicatii de acest tip :

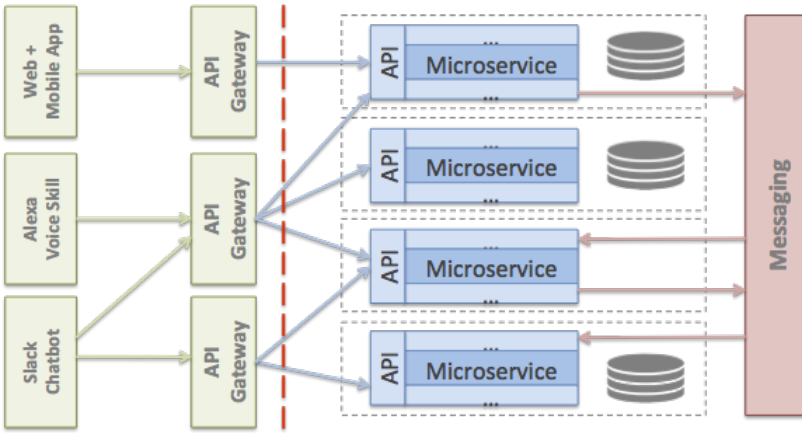

Figura 1.1: Exemplu de arhitectura baza pe microservicii

Capitolul 2

Arhitectura Aplicatiei

In acest capitol va fi prezentata atat schita arhitecturii cu o ilustrare detaliata unde se vor putea observa elementele aplicatiei server (back-end) cu, componentele acesteia, descrierea fiecarei componente in parte, modul lor de functionare, cat si argumentarea folosirii lor.

2.1 Structura generala a aplicatiei

Aplicatia neavand interfata web, arhitectura este orientata server-based ce are ca principale caracterestici doua layere de microservicii care au aproximativ aceeasi functionalitate de calcul al datelor spatiale, dar implementarea si modul de functionare sunt diferite deoarece fiecare dintre acestea cu toate ca preiau aceleasi tipuri de date care sunt pasate catre acestea de o aplicatie server modul in care sunt parsate si trimise inapoi catre utilizator este diferit.

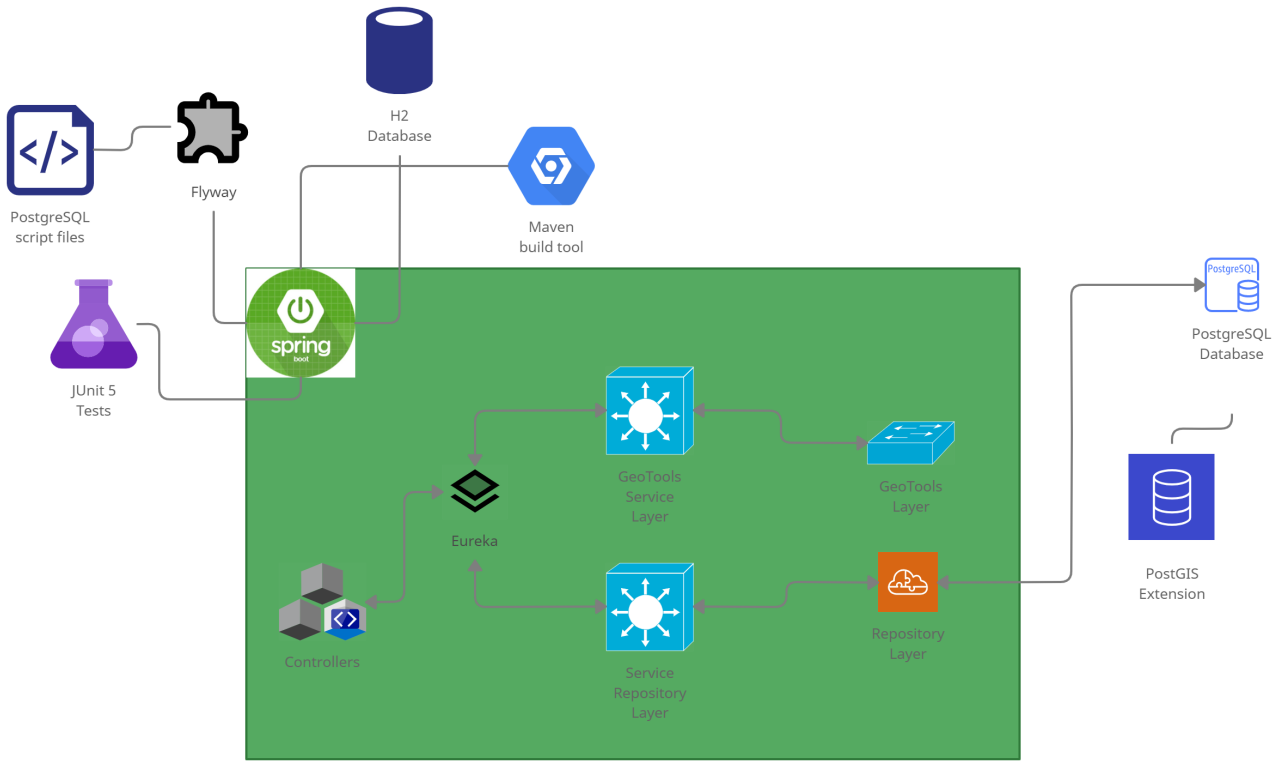

Figura 2.1: Modelul arhitecturii aplicatiei de tip server ce are la baza microservicii cu multiple layere

Aceste date fiind de tip spatial sunt destul de greu de interpretat, iar implementarea ar fi putin dificila, de accea va fi folosita o aplicatie de tip server care, in functie de continutul datelor, paseaza mai departe la microserviciul apelat.

Motivatia alegerii arhitecturii cu microservicii multiple, care foloseste si servicii REST este data de faptul ca este o tehnologie moderna si avansata care ofera multiple implementari si functionalitati. In comparatie cu majoritatea solutiilor existente care folosesc arhitecturi de tip **"Monolith"**, aceasta aplicatie foloseste tehnologii moderne si de actualitate de microservicii.

- Folosirea de **"Aplicatie Server"**, care preia datele de la controllere, le analizeaza si le transmite mai departe catre microserviciul corespunzator care prelucreaza datele si returneaza rezultatul cerut.

- Avantajele microserviciilor par suficient de puternice pentru a convinge unii jucători de mari întreprinderi să adopte metodologia. Comparativ cu structuri de proiectare mai monolitice, microserviciile oferă:

  - **Izolarea erorilor:** aplicatiile mai mari pot ramane in mare parte neafectate de esecul unui singur modul.
  - **Usurinta de intelegere:** cu o simplitate suplimentara, dezvoltatorii pot intelege mai bine functionalitatea unui serviciu.
  - **Implementari mai mici si mai rapide**: baze de cod si scop = implementari mai rapide, care permit, de asemenea, inceperea de a explora beneficiile implementarii continue.
  - **Scalabilitate:** Deoarece serviciile sunt separate, se pot scala mai ușor cele mai necesare servicii la momentele potrivite, spre deosebire de intreaga aplicație. Cand se face corect, acest lucru poate avea un impact asupra economiilor de costuri de performanta.

- Utilizand layere multiple se poate face o distinctie clara intre activitatea de tip web realizata cel mai bine în controler și logica de afaceri generica care nu este legata de web. Putem testa logica de afaceri legata de servicii separat de logica controlerului. Putem să specificam comportamentul tranzactiei, deci daca avem apeluri catre mai multe obiecte de acces la date, putem specifica ca acestea apar în cadrul aceleeasi tranzactii.

2.2 Analiza arhitecturii

In aceasta sectiune se va face o analiza mai detaliata a arhitecturilor folosite in aplicatie, descrierea motivului folosirii acestora, folosinta acestora si modul de operare al aplicatiei.

Intreaga aplicatie va avea ca baza o "Aplicatie Server" numita "Eureka" care la randul ei este alcatuita din multiple componente specifice configurarii de multiple microservicii ce formeaza aplicatia cloud dorita. Aceasta tehnologie este oferita de **Spring** si este numita **Spring Cloud Netflix**.

Componentele vor fi enuntate mai jos explicand modul lor de utilizare, functionalitatea lor cat si cum sunt conectate unele cu celelalte. Pentru crearea acestei aplicatii pe scara larga a fost nevoie de crearea a mai multor instante ale framework-ului **Spring Boot** unde fiecare dintre acestea au rolul sau in alcatuirea aplicatiei in sine. Mai jos vor fi prezentate fiecare dintre aceste module in parte, unde vor fi explicate modul de operare si scopul acestora.

- **WebGISApplicationServer:** Acesta reprezinta modulul principal care tine intreaga aplicatie stabila, monitorizeaza fiecare integrare de modul in parte, maparea acestora corespunzator si setup-ul server-ului. Configurarea acestui server s-a facut la nivel de annotari si configurare de propietati. Pentru accesarea acestui server unde se pot observa toate instantele "Spring" create si un health check al intregii aplicatii se poate accesa link-ul : `http://localhost/8661`

- **WebGISApplicationClientRepository:** Modulul acesta reprezinta unul dintre cele 2 microservicii ale acestei aplicatii. Acesta fiind cel care se foloseste de baza de date "PostgreSQL" pentru extragerea datelor si cu extensia acesteia, "PostGIS", pentru procesarea datelor cu suport spatial. Pentru a putea fi legat de "Aplication Sever" a fost nevoie de o configurare minutioasa a acestuia si crearea unei legaturi stabile cu baza de date. Pe langa aceste configurari de crearea a unei legaturi cu "Eureka" a fost implementata cat si :

  - **Flyway:** este un instrument de migrare a bazelor de date open-source. Favorizează puternic simplitatea și convenția față de configurație. Acesta a fost integrat pentru a putea avea un overview asupra scripturilor ".sql" care se ocupa de crearea de "schemas", tabelelor necesare, datelor de utilizare cat si a extensilor. Acesta ajuta aplicatia in cazul unei migrari sa creeze tot ce ar avea nevoie ca aplicatia sa functioneze optim. In cazul in care ar fi o modificare asupra script-urilor "Flyway" ar opri aplicatia atentionand ca a fost facuta o modificare si nu corespunde cu istoricul acestuia.

In continuare acest microserviciu foloseste design pattern-ul “Repository Pattern“ care este alcatuit din layere multiple :

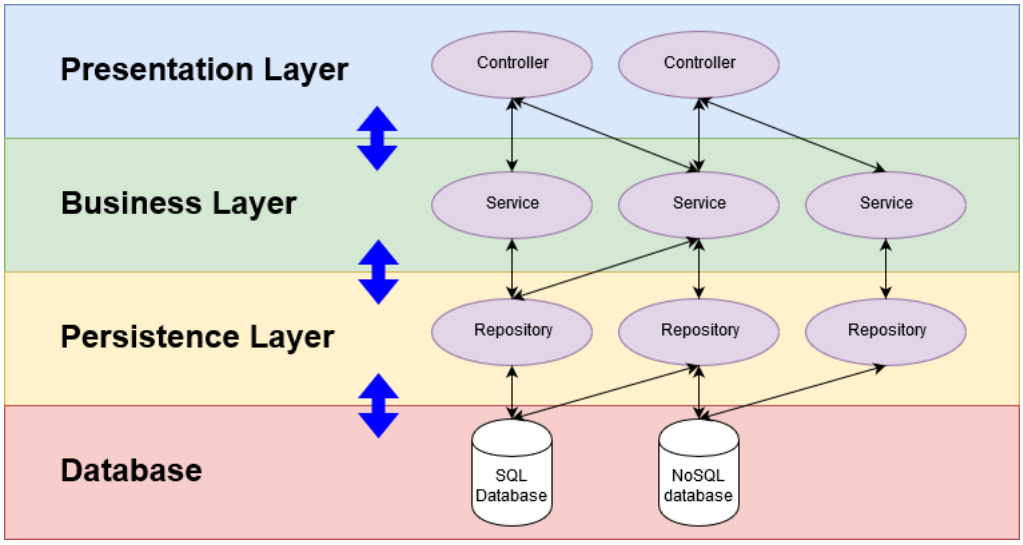

Figura 2.2: Exemplu de implementare a design pattern-ului “Repository Pattern“

- **Service:** Reprezinta layer-ul intermediar dintre “Controller“ si “Repository Layer“ care preia datele de la controller, si de a le da mai departe catre repository pentru a putea realiza prelucrarea datelor.
- **Repository:** Layer responsabil cu comunicarea cu baza de date si manipularea datelor primite din partea clientului. Acesta este legat de baza de date “PostgreSQL“ unde trimite diferite **“DDL Statements“** care fac operatii cum ar fi : extragerea tuturor locatiilor spitalelor,cea mai scurta locatie dintr-un punct (ex: cea mai scurta distanta catre o institutie medicala), afisarea tuturor locatiilor dintr-o zona pe o anumita raza. Pe langa operatiile simple **“CRUD“**, cum baza de date are ca extensie un suport de date spatial **“PostGIS“**, se vor realiza calcule cu aceste date oferind un suport pentru operatii cum ar fi: cea mai scurta distanta catre o anumita institutie, distanta dintre doua locatii etc. Extensia **“PostGIS“** avand propriile sale functii si metode de prelucrare a datelor, ofera sprijin si ajutor in operatiilor cu date de tip spatial. Prin utilizarea annotarii **“@Query“** putem folosii asemenea metode. Un exemplu ar fi :

```
// TransportInstitutionRepository.java
@Query(value = "SELECT buss_stations.id AS
    ↪ id,buss_stations.nume AS nume, "buss_stations.code AS
    ↪ code" + ", buss_stations.latitude AS latitude,
    ↪ buss_stations.longitude AS longitude" + " FROM
    ↪ public.buss_stations buss_stations" ,nativeQuery = true)
List<TransportInstitution> getBusStationLocations();
```

– **Controller:** Reprezinta **Presentation Layer** care se ocupa cu preluarea request-urilor venite din partea de front-web (aplicatie web) si transmiterea mai departe la urmatoarele layere care se vor ocupa cu prelucrarea datelor primite. Acesta este alcatuit din mai multe clase care fiecare reprezinta o functionalitate diferita pentru aplicatia in sine :

  * **TransportLocationController**
  * **MedicalLocationController**
  * **PublicLocationController**
  * **UserController**

  La nivelul acestui layer se realizeaza si validarile obiectelor primite in format JSON si parsate la POJO-urile aplicatiei. In fiecare POJO exista validari la nivelul fiecarui field care verifica corectitudinea informatiilor iar, in cazul in care o conditie nu este indeplinita, acest microserviciu sa arunce o eroare inapoi catre client. Aceste validari se realizeaz cu libraria (proiect separat) numit **Validation** si se foloseste utilizand annotarea “@Valid“ :

```
public List<Institution> getAllLocationsFromZone(@Valid
    ↪ @RequestBody Point point){
    //implementation
}
```

Cum s-a putut observa aplicatia foloseste **Spring Data JPA** care are la baza “CrudRepository“ ce foloseste operatii **CRUD**, astfel permitand utilizarea de “Named Queries“ pentru a trimite comenzi catre baza de date. Acesta annotare are la baza metodele “Java“, asadar se poate lega direct de **Spring Data JPA**.

Pentru aceste “Queries“ s-au creat si utilizat entitati pentru preluarea datelor din baza de date. Aceste entitati sunt simple “POJOS’s“ care sunt annotate cu “@Entity“ pentru a putea lucra cu proiectul separat : **Spring Data JPA**. Mai trebuie precizat faptul ca a mai fost folosita si libraria **Lombok** pentru scrierea a cat mai putin cod. Acesta se utilizeaza folosind annotari precum “@Data“ care creeaza pentru clasa respectiva, “Getters, Setters etc“, cat si alte annotari care ajuta developer-ul sa fie cat mai productiv si eficient.

De asemenea mai trebuie mentionat faptul ca microserviciul este documentat folosind Open API-ul **Swagger**. Folosind annotari corespunzatoare s-au putut documenta fiecare endpoint al aplicatiei cat si POJO-urile utilizate in implementare. **Swagger** fiind un Open API puternic acesta permite prin intermediul url-ului `http://localhost:8183/swagger-ui.html` vizualizarea tuturor endpoint-urilor cat si testarea acestora. Acesta se acceseaza folosind server port-ul microserviciul, astfel putem sa deosebim "ClientRepository" de "ClientGeoTools".

- **WebGISApplicationClientGeoTools:** Modulul acesta reprezinta celalalt microseviciu al aplicatiei. Acesta nu se foloseste de baza de date "PostgreSQL" pentru procesarea de date ci utilizeaza libraria "Open Source" **GeoTools** care foloseste implementari Java pentru prelucrarea si procesarea datelor.

  – **GeoTools** este o bibliotecă de cod Java open source (LGPL) care oferă metode conform standardelor pentru manipularea datelor geospațiale, de exemplu pentru implementarea sistemelor de informații geografice (GIS). Biblioteca GeoTools implementează specificațiile Open Geospatial Consortium (OGC) pe măsură ce sunt dezvoltate. Aceasta librarie este de asemenea "Open Source" oferind codul sursa catre un public "GIT Repository" si de asemenea este "Open Development" oferind sprijin companiilor pentru oferirea de release-uri specifice si stabile.

Ca si celalalt microserviciu acesta se leaga de "Application Server", primul modul prezentat. Configurarea este aceeasi, doar ca acesta trebuie sa aiba un alt nume al aplicatiei cat si un alt server port pentru a nu se suprapune cu celalalt microserviciu pentru ca **Eureka** sa stie sa le diferentieze si sa poata sa faca legaturile necesare fara probleme. Acest microserviciu implementeaza doar acele endpoint-uri care folosesc extensia **PostGIS** folosind **GeoTools**, la fel ca si modulul anterior, se foloseste "Repository Pattern", avand :

– **Presentation Layer (Controller):** Care se ocupa cu preluarea requesturilor venite din partea de front-web (aplicatie web) si transmiterea mai departe la urmatoarele layere care se vor ocupa cu prelucrarea datelor primite. Acesta este alcatuit dintr-o singura clasa care se ocupa de acele endpoint-uri ce pot utiliza libraria externa. Aceste endpoint-uri primind datele identic ca la celalalt microserviciu si rezultatele identice in proportie de 90%.

– **Business Layer (Service):** Reprezinta layer-ul intermediar dintre "Controller" si "Repository Layer" care preia datele de la controller, si de a le da mai departe catre repository pentru a putea realiza prelucrarea datelor.

– **Persistence Layer (Repository):** Este layer-ul principal care se ocupa cu partea de prelucrare si parsare a datelor primite de client. Acesta utilizeaza libraria “GeoTools“ unde calculeaza distanta dintre 2 puncte date, preia locatia unei anumite insitutii (spital, farmacie) cea mai apropiata de un anumit punct dat cum ar fi locatia user-ului, toate locatiile dintr-un anumit punct dat ,dar se poate returna si un anumit tip de locatii cum ar fi : toate scolile, spitalele etc.

Si acest modul se foloseste de simple “POJOS’s“ care sunt annotate cu“@Entity“ pentru a putea lucra cu proiectul separat : “Spring Data JPA“ si sunt utilizate si annotari din proiectul “Lombok“ pentru productivitate si eficienta.

De asemenea si acest microserviciu este documentat cu Open API-ul **Swagger** folosind aceleasi annotari atat pentru controller pentru a documenta endpoint-urile cat si “POJO-urile“ folosite in acesta. Pentru a accesa aceasta documentatie trebuie accesat url-ul `http://localhost:8184/swagger-ui.html`.

Pentru a diferentia microserviciile se vor utiliza porturi diferite.

- **WebGISApplicationFeignClient:** Acesta este unul dintre cele 2 module de **Feign Client** care se ocupa de imbinarea clientului de servicii web cu modulul de microserviciu numit **WebGISApplicationClientRepository**, astfel putem sa legam microserviciul respectiv prin modulul de **Feign**, iar un alt modul **Zuul-Proxy** (despre care se va vorbi la urma) care va face legaturile dintre microservicii si feign client-urile lor respective.Acestea nu fac altceva decat sa paseze mai departe datele catre microserviciul corespunzator.Configurarea unui asemenea client este destul de simpla, tot ce este nevoie este sa oferim un nume distinct cat si un port pentru a putea fi inregistrat de “Eureka“. Un **Feign Client** este alcatuit dintr-un controller, serviciu si “POJO-urile“ ce le utilizeaza.

  - **ClientResourceController:** Clasa aceasta se comporta ca un “RestController“ obisnuit, dar ca resurse foloseste o instanta a unei interfete service numita **ClientServiceFeign** care are ca annotare “$@FeignClient(name = "web-gis-client-repository")$“, unde “name“ reprezinta numele microserviciului pe care il foloseste ca resursa.

    Aceasta configurare de “Resource Controller“ arata asa :

```
@Autowired
public ClientResourceController(ClientServiceFeign
    clientServiceFeign){
        this.clientServiceFeign = clientServiceFeign;
        //resursa de feign client
}

@ResponseBody
@GetMapping("/public/locations/get") // endpoint-ul complet
List<PublicInstitution> getAllPublicLocations(){
    return clientServiceFeign.getAllPublicLocations(); //
        apelarea resursei
}
```

```
11
12 //code continues here
```

– **ClientServiceFeign:** Reprezinta interfata ce este imbinata cu microserviciul "WebGISApplicationClientRepository" si prin intermediul proxy-ului se transmit datele catre acest modul.

Iar acea configurare de "Feign Client" arata asa :

```
1  @FeignClient(name = "web-gis-client-repository") // legatura
   ↪  cu microserviciul
2 public interface ClientServiceFeign {
3
4 @ResponseBody
5 @GetMapping("/public/locations/get")
6 List<PublicInstitution> getAllPublicLocations();
7
8 //code continues here
```

- **WebGISApplicationFeignClientGeoTools:** Este ultimul din cele 2 module de **Feign Client** care se ocupa de imbinarea clientului de servicii web cu modulul de microserviciu numit **WebGISApplicationClientGeoTools**, astfel putem sa legam microserviciul respectiv prin modulul de **Feign**, iar un alt modul **ZuulProxy** care va face legaturile dintre microservicii si feign client-urile lor respective. Ca si modulul "Feign Client" anterior, are aceleasi implementari si configuratii.

  – **ClientGeoToolsResourceController:** Clasa aceasta se comporta ca un "RestController" obisnuit, dar ca resurse foloseste o instanta a unei interfete serviciu numita **ClientServiceFeign** care are ca annotare "$@FeignClient(name = "web-gis-client-geo-tools")$", unde "name" reprezinta numele microserviciului pe care il foloseste ca resursa.

  – **ClientGeoToolsServiceFeign:** Reprezinta interfata ce este imbinata cu microserviciul "WebGISApplicationClientGeoTools" si prin intermediul proxy-ului se transmit datele catre acest modul.

- **WebGISApplicationZuulProxy:** Acesta reprezinta un modul (o instanta de "Spring Boot") care are rolul de a face legaturile dintre microserviciile enuntate si detaliate mai sus si feign client's care ajuta la scrierea de servicii web intr-un mod mai elaborat si mai usor. Configurare este realizata la nivel de propietati si ar arata in felul urmator: fiecare feign client reprezinta un resource service pentru microserviciul lui respectiv care este un client service. Accesarea acestor

microservicii se va realiza pe acelasi port, dar pentru a deosebi ce modul folosim, endpoint-ul de accesare este diferit pentru fiecare : /api/* este pentru “web-gis-client-repository“ iar, /geo-tools/** pentru “web-gis-client-geo-tools“. Aceasta configurare arata in felul urmator :

```
zuul:
  host:
    connect-timeout-millis: 5000000
    socket-timeout-millis: 5000000
  ignoredServices: '*'
  routes:
    geo-tools-resource-service:
      path: /api/geo-tools/**
      serviceId: web-gis-application-feign-client-geo-tools
      stripPrefix: true
    resource-service:
      path: /api/**
      serviceId: web-gis-application-feign-client
      stripPrefix: true
    user-service:
      path: /repository/**
      serviceId: web-gis-client-repository
      stripPrefix: true
    geo-tools-user-service:
      path: /geo-tools/**
      serviceId: web-gis-client-geo-tools
      stripPrefix: true
```

Datele primite vor fi in format **“JSON“**(JavaScript Object Notation) print intermediu protocoalelor **“HTTP“**. Cele mai intalnite protocoale **“HTTP“** sunt urmatoarele:

- **GET:** folosit pentru extragerea de informatii din baza de date.
- **POST:** utilizat pentru salvarea datelor in cadrul bazei de date.
- **PUT:** folosit pentru actualizarea datelor existente.
- **DELETE:** folosit pentru stergerea datelor.

2.3 Validarea si Tratarea Exceptiilor

In aceasta sectiune se va discuta atat despre validarea datelor provenite din partea de front-end (partea web a aplicatiei), acestea venind sub format JSON, cat si despre tratarea exceptiilor care sunt de forma: locatia dupa nume/id nu este gasita, obiectul JSON are parametrii lispa sau incorecti etc.

- **Validari:** acestea au un rol foarte important in asigurarea stabilitatii aplicatiei server cat si de corectitudine a datelor provenite de la client. De pe partea clientului exista posbilitatea sa se trimita un bad HTTP request si datele din baza de date sa fie avariate, in acest sens s-au luat multiple precautii pentru a asigura validitatea datelor. Aceste validari, provenite din pachetul extern **Hibernate** se folosesc de numeroase annotari care puse deasupra field-urilor unui POJO Java, in cazul nerespectarii acelei restrictii acesta va arunca o exceptie corespunzatoare. Mai jos este un exemplu de acest tip de validare:

```
1 @NotNull(message = "The string name may not be null")
2 @NotEmpty(message = "The string name may not be empty")
3 @NotBlank(message = "The string name may not be blank")
4 @Pattern(regexp="^[A-Za-z]*",message = "Invalid Input")
5 private String name;
```

  - **@NotNull:** se asigura ca field-ul nu va avea o valoare null, deoarece in acest caz s-ar putea returna valori corupte.
  - **@NotEmpty:** acesta se asigura ca field-ul nu va fi gol, adica lungimea acestuia sa fie mai mare ca zero.
  - **@NotBlank:** se asigura ca field-ul sa nu fie doar spatii goale sau caractere nevizibile si sa se trimita un query corupt catre baza de date. Cu toate ca tabelele din baza de date nu ar fi afectate si doar s-ar fi returnat empty list sau null, am asigurat sa nu se execute algortimii de parsare a datelor fara sens.
  - **@Pattern:** aceasta annotare se asigura ca field-ul in functie de un regex pattern dat sa indeplineasca aceasta conditie. In cazul de mai sus textul sa contina doar litere.

  Pentru toate aceste annotari, in cazul in care conditia nu este indeplinita se va afisa mesajul custom dat mai sus pentru fiecare annotare in parte. Totusi pentru a face ca aceasta validare sa functioneze este nevoie de inca o annotare speciala care verifica datele in formatul JSON imediat de la primirea prin endpoint-ul respectiv:

```
public List<InstitutionDTO> getAllLocationsFromZone(@Valid
    ↪ @RequestBody Point point){
    return userService.getAllLocationsFromZone(point);
}
```

  - **@Valid:** aceasta annotare activeaza verificarea datelor prin pachetul extern “Hibernate“, folsoind acele annotari exemplificate si explicate mai sus. In cazul unei verificari esuate o exceptie va fi aruncata.

- **Tratarea Exceptiilor:** O excepție este un eveniment care are loc în timpul executării unui program, care perturbă fluxul normal al instrucțiunilor programului. Acestea se manifesta prin aruncarea unui obiect de acest tip numit : **Exception Object** catre “runtime system“. Prin tratarea si manipularea exceptiilor putem sa asiguram flow-ul fluid si neintrerupt al aplicatiei noastre. Validarea si tratarea exceptiilor sunt combinate pentru a putea crea o stransa legatura intre acestea si tratarea celor mai frecvente si daunatoare exceptii. Mai jos vor fi prezentate exceptiile care se pot realiza in interiorul aplicatiei.

  - **Status: 400 Bad Request:** acesta manifestandu-se datorita unui obiect JSON trimis gresit sau incomplet afisand mesajul:

```
{
    "status": "BAD_REQUEST",
    "message": "Operation not permitted.",
    "errors": [
        "Operation not permitted. because something inside the
        ↪  object or the object was : null"
    ]
}
```

  - **Status: 404 Not Found:** acesta status este declansat daca pentru gasirea unei locatii dupa id sau dupa nume nu a fost gasita se va trimite mesajul de mai jos

```
{
    "status": "NOT_FOUND",
    "message": "Object not found",
    "errors": [
        "Object not found because something inside the object
        ↪  or the object was : null or doesn't exist."
    ]
}
```

– **Status: 406 Not Acceptable:** acesta status este declansat daca tipul de institutie trimis nu este gasit/valabil

```
{
    "status": "BAD_REQUEST",
    "message": "Operation not permitted.",
    "errors": [
        "Operation not permitted. because something inside the object or the object was : null"
    ]
}
```

Cu toate acestea, validarea si tratarea exceptiilor ce pot veni printr-un endpoint sunt tratate prin crearea de exceptii custom si crearea unei clase cu o annotare specifica acestui tip de situtatie. Pentru afisarea mesajelor de exceptii se folosete o clasa custom care preia doar informatiile necesare. Aceste clase custom sunt de forma:

```
public class ConstraintViolationExceptionCustom extends RuntimeException {
    public ConstraintViolationExceptionCustom(){
        super("Bad Object Request.");
    }
}
```

Iar clasa principala de prinderea si afisarea mesajelor de eroare este de forma urmatoare:

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ResponseStatus(value = HttpStatus.NOT_FOUND, code = HttpStatus.NOT_FOUND)
    @ExceptionHandler(NotFoundException.class)
    public ResponseEntity<Object> notFoundException(RuntimeException exception) {

        String error = exception.getMessage() + " because something inside the object or the object was : " +
        exception.getCause() + "" +
                " or doesn't exist.";

```

```
11          APIError apiError =
12                  new APIError(HttpStatus.NOT_FOUND,
                    ↪  exception.getLocalizedMessage(), error);
13          return new ResponseEntity<>(apiError, new HttpHeaders(),
            ↪  apiError.getStatus());
14      }
15
16      ...
```

Annotarea "@RestControllerAdvice" se pune deasupra clasei pentru a fi instintata in contextul "Spring" ca si clasa de tratare a exceptilor pentru endpoint-uri iar, "@ExceptionHandler(NotFoundException.class)" este folosit pentru a specifica clasei ce tip de eroare asteptam (in interiorul parantezelor se va specifica tipul de exceptie la care ne asteptam. De retinut este faptul ca, pentru fiecare microserviciu si feign client-ul respectiv al acestuia are o asemenea configuratie de tratare a exceptilor.

2.4 Motivare tehnologii alese

In aceasta sectiune se va vorbi in special de motivatia alegerii tehnologiilor alese, cat si avantajele acestora. Aplicatia propriu-zisa este dezvoltata cu ajutorul unui framework "Spring Boot", se va specifica limbajul de progrmare folosit la baza acestuia, acesta fiind Java. Spring Boot este un micro-cadru open-source întretinut de o companie numita Pivotal. Ofera dezvoltatorilor Java o platforma pentru a incepe cu o aplicatie de "Spring", configurabila automat, de productie. Cu acesta, dezvoltatorii pot incepe rapid fara a pierde timp la pregatirea si configurarea aplicatiei lor de Spring. Unele dintre avantajele acestui framework fiind urmatoarele:

- **Flexibilitate:** Spring suport atat configurare legacy pe baza de XML cat si "Java-based annotations" care este de noutate si te ajuta pentru configurarea de "Spring Beans". Acesta putand sa fie rulat cu orice tip de configurare permite crearea de aplicatii enterprise foarte usor.
- **Standalone:** Framework-ul Spring vine la pachet cu un lightweight container. Acesta poate fi activat fara se se foloseasca un web server sau application server. Cu toate acestea aplicatia creata foloseste o versiune lite de web server Apache Tomcat care este foarte rapid si stabil.
- **Cadrul web bine conceput:** Spring are un framework MVC foarte bine construit care aduce o varietate de alternative la un web framework legacy si de asemenea permite integrarea foarte usoara a altor pachete pentru extinderea si imbunatatirea aplicatiei server web.

In concluzie, Spring este un framework care a ajuns la maturitate, este intretinut si actualizat cu grija. Este foarte rapid in ceea ce priveste atat performanta acestuia cat si viteza de productie a unei aplicatii server web.

Capitolul 3

Facilitati Aplicatie

Pentru prezentarea facilitatilor aplicatiei, am creat o diagrama **“Use Case“**(diagrama a cazurilor de utilizare) care prezinta actiunile principale ale aplicatiei.

Cum lucrarea de licenta este axata mai mult pe partea de server-based, diagrama prezentata este construita pentru a ilustra cazurile de utilizare din perspectiva server-ului.

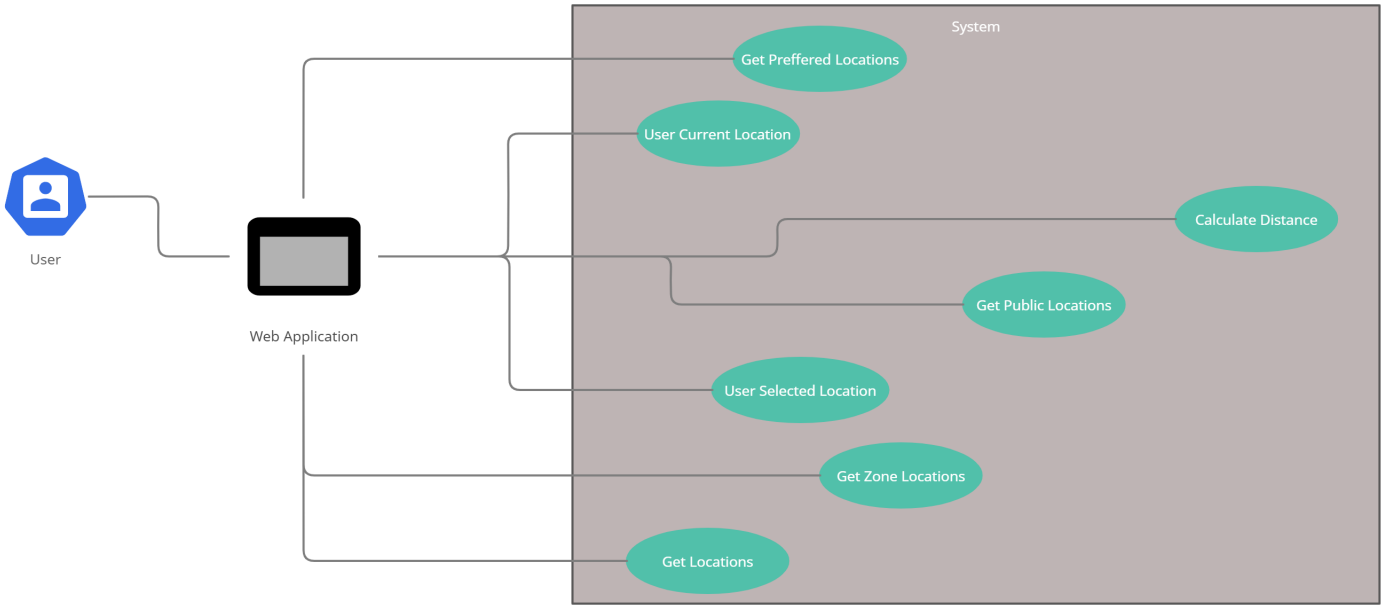

Figura 3.1: Diagrama **Use Case** unde sunt exemplificate cateva din cazurile de utilizare ale aplicatiei

Mai jos vor fi explicate operatiile efectuate de cazurile de utilizare prezentate mai sus:

- **User Current Location:** Acest use case va returna locatia curenta a utilizatorului.
- **User Selected Location:** Acesta va returna o locatie data de utilizator.
- **Get Zone Locations:** Use case-ul acesta este putin mai complex deoarece va returna in baza unei locatii date, toate zonele de interes dintr-o anumita zona.
- **Get Locations:** Acest use case va returna locatii in functie de preferintele clientului pe o anumita raza.

- **Get Preferred Locations:** Va oferi utilizatorului locatii in functie de o anumita categorie aratand drumul cel mai scurt dintr-o anumita zona selectata de acesta.
- **Calculate Distance:** Asa cum sugereaza si numele, in functie de doua locatii date de utilizator, aplicatia va returna distanta dintre acestea cat si alte date utile pentru acesta.

Bibliografie

[1] John Carnell *Spring Microservices in Action Second Edition.* Manning Shelter Island 2017

[2] `https://flywaydb.org/documentation/`

[3] `https://docs.spring.io/spring-cloud/docs/current/reference/html`